

ATTACHMENT 11

From: Phillip Remaker <remaker@cisco.com>
To: Carl Schaefer <schaefer@cisco.com>;Murali Venkateshaiah <muraliv@cisco.com>
CC: Chao-Li Tarng <ctarng@cisco.com>;parser-police@cisco.com <parser-police@cisco.com>
Sent: 1/12/1999 7:05:19 PM
Subject: Re: Parser review guideline?

This is not exactly a comprehensive guideline of parser philosophy, but a clarification of finer points for experienced engineers 8-)

Your milage may vary.
Parser-Police Manifesto, version 1.7
"To Serve And Protect"
Author: remaker@cisco.com
This document is not an offical Cisco procedure
It is for reference use only.

PURPOSE

Parser-police is an alias for software professionals in Cisco to propose command line additions and get feedback from fellow engineers with experience to insure consistency, usability, and friendliness of the configuration interface to Cisco IOS.
It is not for discussion technical coding or parse chain issues.
Send those questions to "parser-questions."

AUTHORITY

The official design procedures for IOS *require* that certain classes of changes be cleared with parser-police, but in practice parser-police has no formal "clearing" criteria. It is a forum which has a history of preventing bad things (tm) from getting into the command line interface (CLI). Most of the serious abominations in IOS did not pass the parser-police.

However, since it has no specific authority, parser-police derives its authority by having good answers, level-headed discourse, and a history of successes. This does not mean it's okay to ignore comments. Generally, the people doing the reviews on parser-police have more IOS CLI experience than the people behind the submission, so this should weigh heavily in evaluating feedback on the list.

SUMBMISSION

All command line interface additions, whether config, exec, or whatever, MUST be sent to parser-police for review.

* The submitter should send the proposed syntax BEFORE writing any code, to prevent rewrites for syntax changes. "I already coded it" is not an acceptable excuse for poor syntax.

* The submitter should submit plain text, not pointers to specs, MIME attachments, diffs or parser macros.

* The submitter should make the message as brief as possible, and include the command syntax, what it does, and a brief reason for the choice if it seems that the command will be controversial. If there were any choices that you anticipate being suggested which you already know that you will reject, you may want to include that information and reason for rejection to pre-empt replies.

* The submitter should include a pointer to a functional spec or DPTS if application. This inclusion is IN ADDITION TO and not in place of a summary text description as described above.

* The submitter should also include a list of target IOS versions that will be getting the new command, so that the good folks in documentation can be sure that your handiwork becomes canon and not apocrypha. Far too often, a useful syntax addition becomes an undocumented command because nobody let documentation know about it.

* The submitter should specify a DEADLINE for comments, so that the discussion is bounded. Seven days is generally accepted. A longer time is better. Shorter times are acceptable only in the case of high priority bugs which require adding a new command.

ANSWERING

Answer submissions with respect. Finger pointing, value judgements, and summary dismissals are uncalled for. If there is a problem, state clearly what the problem is and OFFER AN ALTERNATIVE. If you can cite examples in other parts of the parse tree, please do so.

In general, your reply should include parser-police. You can privately send mail when it is a simple concurrence or a point of information that may not be of interest to the whole group.

Please note that precedence is not always automatic grounds for acceptance of a particular syntax, since there are a number of places in the parse tree where bad syntax has crept in (usually due to the fact that the syntax was unreviewed by parser-police!!).

If you cannot offer an alternative, say so. Remember: since parser-police has no specific enforceable authority per se, the practice of conducting useful discourse ensures that the parser-police alias will remain relevant. If you have a personal issue with the submitter, take it to software-flame, off line or through your management chain.

NO ANSWER

Silence is usually a sign that the command is OK. If there is no reply after a week, please repost the message saying that you are going ahead with the proposed syntax.

NO CONSENSUS

If the discussion deadlocks, and no resolution can be reached, the submitter must post a summary of the discussion and the final decision. In practice, the submitter has the choice of how to deal with feedback, but protocol dictates that the most appropriate and professional course of action is for the submitter to take responsibility to work for consensus. The submitter and parser-police are JOINTLY responsible for developing a mutually acceptable syntax. The cumulative expertise of the parser-police is on what constitutes "good" syntax. The submitter will generally be expert on the particular application, knowledge of the customer, etc. Each is responsible for educating the other in order to avoid review deadlock.

If anyone receiving the submitter's "final decision" e-mail still finds the final choice unacceptable, resolution should be pursued through appropriate management chains, which vary widely by specific situation. Sometimes it will be the submitter's manager, or the division director, or even the VP. Your professional judgement is your guidance here.

SYNTAX DESIGN GUIDELINES

1) Think extensible. If you add a command, try to envision if more similar commands that may be added, and structure the parse tree not to have 'dead ends'.

BAD dnsix-dmdp
GOOD dnsix dmdp

See how dnsix-dmdp, as a top level command, precludes any other

dnsix related commands without making a new top level command (Which was done later, with "dnsix-nat"). If the top level keyword had been "dnsix", future dnsix settings could have been added to the parse chain gracefully. This also illustrates an instance of the hyphens-in-commands controversy, discussed below.

2) Hyphens should be avoided if they indicate sub-keywords are warranted. Similarly, you should not be overzealous to eliminate hyphens. However, multi-hyphens are usually a sign that keywords are too long and the concept should be re-thought. All multi-hyphen commands should be analyzed to see if they can be split into a more extensible parse chain.

BAD debug [isdn-q931 | isdn-q921]
 GOOD debug isdn [q921 | q931]
 This put logical isdn debugs under one umbrella.

GOOD ip forward-protocol spanning-tree
 BAD ip forward protocol spanning tree
 This creates unneeded subsets. "forward-protocol" is one concept and not extensible, as is "spanning-tree"

BAD exec-banner, motd-banner, access-banner
 GOOD banner [access | motd | exec]
 This logically groups all banner related commands together.

BAD isdn not-end-to-end [56|64]
 GOOD isdn speed [56|64] incoming
 The not-end-to-end was a bad hack which is still in the code, which had a specific application. The command REALLY forced a speed lock regardless of the switch info. The command should have been named as such. Also, it only applies to inbound calls. In fact, it would be useful to someday have it for outbound as well, hence the final extension.

BAD all-incoming-calls-v120
 GOOD force-bearer v120
 The "all-incoming-calls-v120" was intended to treat all inbound calls as v120. This really just forced the treatment of the call as V120 regardless of isdn bearer info. By using a command like "force-bearer", this functionality could readily be extended to other bearer types in the future.

3) Enabling and disabling functions

BAD ppp multilink enable ppp multilink disable
 GOOD ppp multilink no ppp multilink
 The "enable" or "disable" as keywords should be deleted. Instead, the command should stand on its own to indicate something being enabled, or with a "no" prefix if it is being disabled. Note that it is okay to have the "no" form of a command appear in the configuration when disabling something that's on by default, e.g. "no ip routing".

BAD compress use-stac
 BAD compress do-stac
 GOOD compress stac
 The configuration should describe the behavior, not command it.

4) Watch for collisions. Since the parser looks for smallest unique match, be on the lookout for adding an obscure keyword that conflicts with a common one. This is also an argument for going deep, not wide with commands as requested in item (2).

5) The fact that a badly designed command in IOS exists does not constitute justification for another badly designed command. It is a sad fact that we have gruesome hacked commands in a number of areas that need fixing. We try to weed out over time, so the last thing we need is more entropy.

6) When naming a command, try to pick names that would be familiar to people in the industry. For example, "ip mtu 576" is better than "ip maximum-transmission-unit 576" since MTU is an accepted industry acronym. However, where the industry has not settled on a universal parlance, the longest formal name is probably better.

7) Do not use code names in commands. "debug whizzy-asic" or "debug walamazoo" will not be very useful to customers. If you need them and you SWEAR customers will not use them, make them hidden PROVIDED THAT you make sure the debug commands do not collide with common debug commands.

8) NEVER use underscores. Use dashes. This is a purely aesthetic thing, but it is important to be consistent. Some underscore commands have been jammed into the code (b_channel, and others) and should be weeded out as soon as possible.

9) Don't hide a command that will be useful for debugging. The way to debug robbed bit signalling in AS5200 is so ass-backwards and convoluted that I can scream. It requires 3 steps to do, when in fact it should be a simple "debug" command. The argument was that it was a hack added in for debugging that should never be seen/used by customers. In fact, CE uses it tens of times, daily. Fundamentally: If there is a process that will output state information on a real time basis, tracking changes of traffic as it occurs, that should be a DEBUG command, using all of the debug protocol (eg, buginf()).

10) Commands should tend to be self-explanatory so that a relatively knowledgable user can figure out the command function from the command and on-line help without having to scurry off to the manuals. What constitutes "self-explanatory" will vary by your target audience, so be prepared to defend that point. While a non-ATM user may find the command "forward-peak-cell-rate-clp1" offensively complex, the point can be made that this will be the only acceptable syntax for the ATM community based on the vocabulary and culture of that user group.

CHANGING SYNTAX

Changing an existing syntax is ususally a bad idea. Once customers are already using a certain syntax, changing the syntax will frequently do more harm than good. There are at least four reasons. **First, customers are trained on and familiar with existing syntax.** Second, customers frequently change IOS versions, sometimes jumping up or down a major revision level. **Having portions of the configuration be unrecognized could cause catastrophic failures.** Third, customers may boot a text file of the config from a server, and may not update that text version to accomodate new syntax. Similar catastrophic failures due to unrecognized commands is possible. And fourth, the fact that some platforms have a ROM based bootloader that may be several revs down would cause syntax-changed config files to generate disturbing, though innocuous, error messages.

For the above reasons, we have many examples of poor syntax that exist in the IOS which remain there indefinitely. However, there are instances where egregious misjudgements in command structure need to be corrected in order to add new functionality to the IOS or to reduce the number of calls to the technical assistance center.

The following chart is a guideline of how to execute a syntax change gracefully, with a minimum of disturbance. Release "0" is the major release where the change. -1 is the one before, etc. A release, in this case, is defined as a major release, eg, 11.2 to 11.3, 11.3 to 12.0, 12.0 to 12.1.

-1 OLD OLD OLD OLD

0 OLD+NEW OLD+NEW OLD OLD+NEW

1 OLD+NEW NEW NEW NEW

2 NEW NEW NEW NEW

This chart is a guideline. Very serious changes may call for a longer cycle. For example, when the command "address" was changed to "ip address," The old command continued to be accepted for 3 major releases since very near 100% of all Cisco customers used the command.